Replacing Object Oriented Programming Features through Aspect Oriented Programming with Crosscutting Concerns

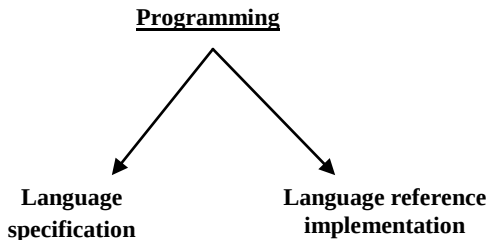
Ravi Kumar
Ph.D. Scholar, MMICT& BM
Maharishi Markandeshwar University
Mullana(Ambala)-133207, Haryana,India

Munishwar Rai, PhD
Associate Professor, MMICT& BM
Maharishi Markandeshwar University
Mullana(Ambala)-133207, Haryana,India

ABSTRACT
Aspect-oriented programming (AOP) has been introduced as a potential programming approach for the specification of nonfunctional component properties of a system. Thus AOP and especially AspectJ (general purpose aspect oriented language) is assessed from the component reuse point of view. We examines the use of the language, as well as its features . It lays out a common crosscutting problem to illustrate the general syntax of the traditional AspectJ language.

Keywords
Aspect Oriented Programming**,** Reusability, AspectJ .

1. INTRODUCTION
An aspect-oriented programming system (AOP) is a software system that is a realization of the aspect-oriented programming methodology [3].



An explicit definition of the syntax and semantics of the language is called language specification. A software application that can translate code written in the language into an executable form is called language reference implementation.

A complete AOP system supports the following fundamental concepts [1, 4-6]:

- Join points: Identifiable points in the execution of a system.
- Pointcut: A construct for selecting join points.
- Advice: A construct to introduce or alter execution behavior .
- Static crosscutting: constructs for altering the static structure of a system .
- Aspect: A module to express all crosscutting constructs .

Fig. 1 shows the central concepts in each of the three programming approaches such as AOP,OOP ,POP and how they are related to each other[8][12].

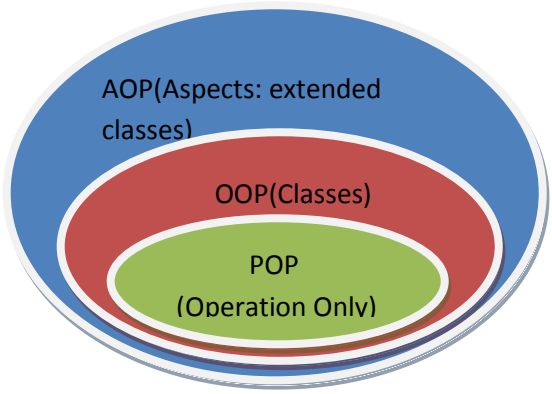


Fig 1: The Relationship between POP, OOP and AOP

1.1 Comparison of OOP and AOP
Table 1.

Characteristics	OOP	AOP
Code unit	Class: Code unit that encapsulates methods and attributes[11].	Aspect: Code unit that encapsulates pointcuts, advice, and attributes.
Entry point	Method signatures: Define the entry points for the execution of method bodies.	Pointcut: Define the set of entry points in which advice is executed.
Implementation	Method bodies: Implementations of the primary concerns.	Advice: Implementations of the crosscutting concerns.
Conversion	Compiler: Converts source code into object code.	Weaver: Instruments code (source or object) with advice.

AspectJ is the first complete AOP system which is also the best and most widely used AOP system. The initial development of AspectJ was the work of a team at Xerox PARC, led by Gregor Kiczales. He also coined the terms

"crosscutting" and "aspect-oriented programming". After a few releases, Xerox donated AspectJ to the free software community at http://eclipse.org. A few years later another AOP system (AspectWerkz) merged with AspectJ, adding features, such as annotation based syntax. At the moment, AspectJ has an alternative implementation(AspectBench), used for experimenting with new features and optimizations .

2. ASPECTJ IMPLEMENTATION IN AOP

AspectJ is an extension to the Java programming language that adds AOP capabilities to Java. AspectJ provides three weaving mechanisms or AspectJ is a three-step process (see Fig. 2)[17]:

1. Compile classes
2. Compile aspects
3. Weave aspects into classes to produce the final binary files

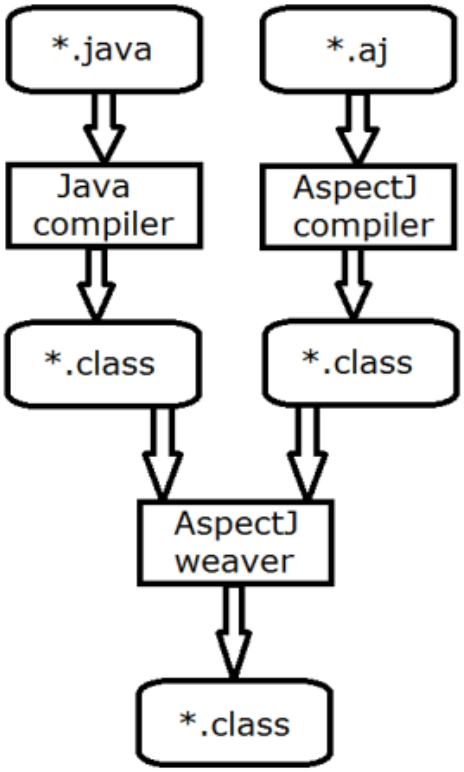


Fig. 2 Three-step process

"AOP is an approach to software development that combines generative and component–based development" [9]. Fig. 3 illustrates the conversion of an Object Oriented program to Aspect Oriented and how a concern is modularized[10].

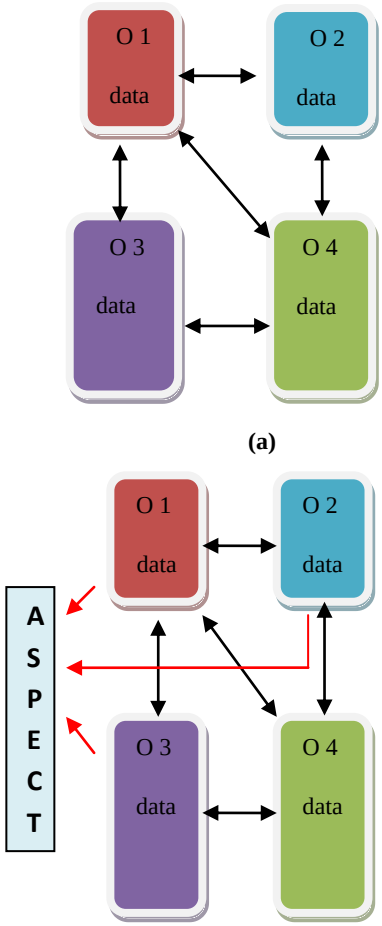


Fig. 3 Modularizing system concerns

AOP complements OOP by providing a different way of thinking about program structure.The key unit of modularity in OOP is the class whereas in AOP the unit of modularity is the aspect. Fig. 3 (a) describe the object oriented program. A program can be categorized into four objects like O1,O2,O3,O4 .Each object could accessed by a class but in AOP we make an aspect as shown in Fig. 3(b) .Through aspect we make more modular of a program. Here aspect is central unit of AspectJ program like a class. Aspect allows the user to understand each element of the system by knowing only its concern and without the need to understand other module. Main concern in a program are identified and implemented as aspects. They are then weaved into the appropriate places in the program by a weaver which is compilerof an aspect language[2].

AOP for implementing AspectJ will certainly enhance software quality in many ways like[7]:

- Clear responsibilities for individual modules : In AOP, AspectJ code deals with the same aspect in one module avoiding the redundancy of crosscutting concerns.

- Consistent implementation : AspectJ provides consistent implementation by having each aspect handled once.
- Improved reusability : AOP isolates core concerns from the crosscutting ones, enabling more mixing and matching, and therefore improving the overall reusability in both modules.
- Improved skill transfer : The concepts of AOP are reusable and transferable. Therefore, developers training time and cost will be minimized even if they need to learn more than one language. This is because core concerns and design patterns are universal.
- System-wide policy enforcement : AOP allows programmers to enforce a variety of contracts and provide guidance in following "best" practices by creating reusable aspects.
- Logging-fortified quality assurance : AOP enables quality-assurance persons to attach the bug paper with its log, easing the reproduction of the behavior by the developer.
- Better simulation of the real world through virtual mock objects : AOP makes the difficult and cumbersome testing process easier without the need to compromise the core design for testability.
- Nonintrusive what-if analysis : AOP does not waste time and space by checking whether functionality is needed by running what-if analysis every time before changing the system behavior.

3. COMPARISON OF JAVA AND ASPECTJ THROUGH EXAMPLE

Join point, point-cut ,advice ,introduction ,point-cut designator are the points through which we can link the AspectJ and Java(non aspect) code[14]. Some join points examples are method call, method execution, and method reception. AspectJ defines different types of point-cut designators that can identify all types of join points [13]. In Fig.4 the point-cut prime operation at statement 12 picks out each call to the method isprime() of an instance of the class prime, where an int is passed as an argument and it makes the value to be available to the point-cut and advice[16]. Using advice we can define certain code to be executed when a point-cut is reached [13]. After advice at statement 15 runs after each join point picked out by the point-cut primeoperation and before the control is return to the calling function. The before advice at statement 13 runs before the join point picked out by the point-cut primeoperation. Introduction allows an aspect to add methods, fields or interfaces to existing classes. A point-cut designator simply matches certain join points at runtime. In Fig.4 the point-cut designator *Call (boolean prime.isprime(int)*, at statement 12 matches all the method calls to factorial from an instance of the class *prime* [15].

4. CONCLUSION

Clearly, there is a need for development environments that support the efficient creation of applications that use modern execution systems. This has been the goal of a continuing research effort over the last several years. The previous focus has been on using component-based ideas to develop a programming model and associated framework to support such a development approach. AspectJ slowly but surely gains popularity and the number of big projects that are using it has increased. AspectJ is also very popular in the academic setting. Researchers often use it for their research in the area of AOP, for example in software design optimizations. AOP with AspectJ has potential to become very popular in the near future because it is easy to use and very powerful.

Non aspect code

```
Import java.util.*;
public class prime
{
private static int n,count=0;
1. public static void main(String args[])
  {
2. n=Integer.parseInt(args[0]);
3. if(isprime(n))
  {
4. System.out.println("IS PRIME");
  else
5. System.out.println("IS NOT PRIME");
  }
 }
6. public static boolean isprime(int n)
  {
7. for(int i=2; i<=n/2; i++)
  {
8. if(n%i == 0)
{
  Count++;
  break;
  }
  }
  If(count==0)
  {
9. return false;
  }
  Else
{
10. return true;
  }
        }
```

Aspect code

```
11. public aspect PrimeAspect
  {
12. public pointcut primeoperation(int n): call (boolean prime.isprime(int) && args(n);
13. before (int n): primeoperation(n)
  {
14. System.out.println("Testing the prime number for " +n);
  }
15. after(int n) returning (boolean result): primeoperation(n)
  {
16. system.out.println(showing the prime status for" + n);
  }
  }
```

Fig. 4 AspectJ program to test a number is prime or not

5. REFERENCES

[1] R. Laddad, AspectJ in Action, 2010 Enterprise AOP with Spring, Manning Publications.

[2] Munishwar Rai, Rajender Nath & Jai Bhagwan, International Journal of Engineering and Innovative Technology(IJEIT), Volume 3, Issue 5, November

2013,PP.309-11," A Cluster Based Reusability Model with Reference to Aspect Mining ."

[3] http://www.eclipse.org/articles

[4] R. Miles, O'Reilly Media, 2004,AspectJ Cookbook.

[5] A. Colyer, A. Clement, G. Harley and M. Webster, Addison-Wesley Professional, 2004,Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and Eclipse AspectJ Development Tools.

[6] J. D. Gradecki and N. Lesiecki, Wiley, 2003, Mastering AspectJ: Aspect- Oriented Programming in Java.

[7] R. Laddad, IEEE Software, 2003, vol.20, no. 6, pp. 90-91. "Aspect-oriented programming will improve quality."

[8] J. Viega, J. Vuas, IEEE Software, 2000, vol.17, no.6, pp. 19-21. "Can aspect-oriented programming lead to more reliable software?,"

[9] Sommerville, 8 ed, 2007, I. Software Engineering,.

[10] Daniela Gotseva and Mario Pavlov**,** IJCSI vol. 9,Issue 5, No.1., 2012, **"**Aspect-oriented programming with AspectJ"

[11] Sk. Riazur Raheman, Amiya Kumar Rath, Hima Bindu M , IJRITCC ,vol.2,Issue:2,pp.249-259,2014, "Dynamic Slice of Aspect Oriented Program: A Comparative Study"

[12] Heba A. Kurdi, IJACSA,vol.4,No.9,2013, "Review on Aspect Oriented Programming"

[13] Mohapatra D. P. et. al., , informatica 32, 261-274, 2008, Dynamic Slicing of Aspect-Oriented Programs.

[14] Abhishek Ray et. al., International Journal of Software Engineering and Its Applications, Vol. 7, No. 1, January, 2013, An Approach for Computing Dynamic Slice of Concurrent Aspect-Oriented Programs.

[15] Gregor Kiczales et. al., published in proceedings of the 15th European Conference on Object Oriented Programming, pages 327-353, 2001, An Overview of AspectJ.

[16] Jyri Laukkanen , seminar paper, UNIVERSITY OF ELSINKI , 2008, Aspect-Oriented Programming.

[17] http://www.eclipse.org/aspectj